

# Randomized Incremental Construction for the Hausdorff Voronoi Diagram of point clusters<sup>\*</sup>

Elena Khramtcova and Evanthia Papadopoulou

Faculty of Informatics, Università della Svizzera italiana (USI), Lugano, Switzerland  
 {elena.khramtcova, evanthia.papadopoulou}@usi.ch

**Abstract.** This paper applies the randomized incremental construction (RIC) framework to computing the Hausdorff Voronoi diagram of a family of  $k$  clusters of points in the plane. The total number of points is  $n$ . The diagram is a generalization of Voronoi diagrams based on the Hausdorff distance function. The combinatorial complexity of the Hausdorff Voronoi diagram is  $O(n + m)$ , where  $m$  is the total number of *crossings* between pairs of clusters. For *non-crossing* clusters ( $m = 0$ ), our algorithm works in expected  $O(n \log n + k \log n \log k)$  time and deterministic  $O(n)$  space. For arbitrary clusters ( $m = O(n^2)$ ), the algorithm runs in expected  $O((m + n \log k) \log n)$  time and  $O(m + n \log k)$  space. When clusters *cross*, bisectors are disconnected curves resulting in disconnected Voronoi regions that challenge the incremental construction. This paper applies the RIC paradigm to a Voronoi diagram with disconnected regions and disconnected bisectors, for the first time.

## 1 Introduction

Given a family  $F$  of clusters of points in the plane, the *Hausdorff Voronoi diagram* of  $F$ ,  $\text{HVD}(F)$ , is a subdivision of the plane into maximal regions such that points in a single region have the same nearest cluster. The distance between a point  $t$  and a cluster  $P$  is the *farthest distance* between  $t$  and  $P$ ,  $d_f(t, P) = \max_{p \in P} d(t, p)$ , where  $d(\cdot, \cdot)$  denotes the Euclidean distance between two points. The farthest distance equals the Hausdorff distance between  $t$  and  $P$ , hence, the name of the diagram. Let  $k$  denote the number of clusters ( $k = |F|$ ) and let  $n$  denote the total number of points in all clusters ( $n = |\cup F|$ ); no two clusters may have a common point. We assume that each cluster equals its convex hull, as only vertices on a convex hull can be relevant to the Hausdorff Voronoi diagram.

The Hausdorff Voronoi diagram was first considered by Edelsbrunner et al. [10] under the name *cluster Voronoi diagram*. The authors showed that its combinatorial complexity is  $O(n^2 \alpha(n))$ , and gave an algorithm of the

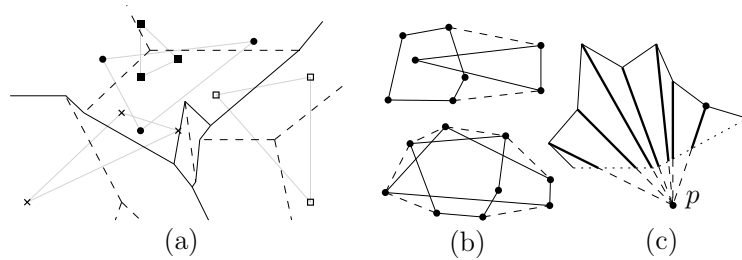
<sup>\*</sup> Research supported in part by the Swiss National Science Foundation (SNF), project 20GG21-134355, under the ESF EUROCORES program EuroGIGA/VORONOI, and project SNF 200021E-154387.

same time complexity to compute it, where  $\alpha(n)$  is the inverse Ackermann function. These bounds were later improved to  $O(n^2)$  [14]. This algorithm is optimal in the worst case, however, not when the complexity of the diagram is subquadratic. When the convex hulls of the clusters are disjoint [10] or just *non-crossing* [14] (see Def. 1), the combinatorial complexity of the diagram is  $O(n)$ . The Hausdorff Voronoi diagram is equivalent to the upper envelope of a family of lower envelopes of an arrangement of hyperplanes in  $\mathbb{R}^3$  (each lower envelope corresponds to a cluster) [10].

**Definition 1.** *Two clusters  $P$  and  $Q$  are called non-crossing, if the convex hull of  $P \cup Q$  admits at most two supporting segments with one endpoint in  $P$  and one endpoint in  $Q$ . If  $P \cup Q$  admits more than two such supporting segments, then  $P$  and  $Q$  are called crossing (see Fig. 1b).*

The combinatorial complexity (size) of the Hausdorff Voronoi diagram is  $O(n + m)$ , where  $m$  is the number of *crossings* between pairs of crossing clusters (see Def. 2), and this is tight [13]. The number of crossings  $m$  is bounded from above by (half) the number of supporting segments between pairs of crossing clusters. In the worst case,  $m$  is  $O(n^2)$ . There have been several algorithms to compute the Hausdorff Voronoi diagram for non-crossing or arbitrary clusters, see e.g., [9,13,14], however, their time or/and space complexities have not been satisfactory.

For non-crossing clusters, the Hausdorff Voronoi diagram is an instance of *abstract Voronoi diagrams*. The bisector of two clusters can have complexity  $\Theta(n)$ , thus, directly applying the randomized incremental construction for abstract Voronoi diagrams leads to an  $O(n^2 \log n)$  algorithm, and this is not easy to overcome, see [5]. When clusters are crossing, their bisectors are disconnected curves [14], and do not satisfy the basic axioms of abstract diagrams. Thus, the Hausdorff diagram does not comply with the recent extension of abstract Voronoi diagrams with disconnected regions [2], and the relevant construction algorithm is not applicable.



**Fig. 1.** (a) HVD; (b) Non-crossing clusters (above) and clusters with 2 crossings (below); (c) visibility-based decomposition

Recently, we presented a randomized incremental algorithm to compute the Hausdorff Voronoi diagram of a family of  $k$  non-crossing clusters, based on point location in a hierarchical dynamic data structure [5]. The expected running time of this algorithm is  $O(n \log n \log k)$  and the expected space complexity is  $O(n)$ . This approach, however, does not generalize to arbitrary clusters, even if their number of crossings is small. Note that clusters with a small number of crossings is the case of interest to our motivating application (explained below).

In this paper, we consider the randomized incremental construction (RIC) framework introduced by Clarkson et al. [6,7] addressing all cases of point clusters in the plane. The RIC approach to compute a Voronoi diagram, inserts sites (or objects) one by one in random order, each time recomputing the target diagram (see, e.g., [3]). The diagram is viewed as a collection of *ranges*<sup>1</sup>, *defined* and without *conflicts* with respect to the set of sites inserted so far. To update the diagram efficiently, a *conflict* or *history* graph is maintained. Maintaining the conflict or the history graph efficiently after each update is crucial for the performance of the algorithm. This is difficult for the Hausdorff diagram, even in the case of non-crossing clusters, because of the special features of this diagram such as sites of non-constant complexity, sites that are not necessarily contained in their regions, and empty Voronoi regions. Furthermore, crossing clusters create disconnected Voronoi regions.

In this paper, we first consider non-crossing clusters, when the complexity of the diagram is  $O(n)$ , and then extend our results to arbitrary clusters with  $m$  crossings, where the complexity of the diagram is  $O(n + m)$ . For non-crossing clusters, our algorithm runs in expected  $O(n \log n + k \log n \log k)$  time and deterministic  $O(n)$  space. In comparison to [5], the construction is considerably simpler, complementary, and it slightly improves the overall time complexity; more importantly, it extends to arbitrary clusters, unlike [5]. We give the construction for both a *conflict* and a *history graph*, where the latter is an on-line variation of the algorithm. Allowing clusters to cross adds an entire new challenge to the construction algorithm: a bisector between two clusters consists of multiple components, and one Voronoi region may disconnect in several faces. This disconnection may happen at any time during the incremental algorithm. We show how to overcome the challenge of disconnected regions on a conflict graph and derive an algorithm whose expected time and space requirements are respectively  $O((m + n \log k) \log n)$  and  $O(m + n \log k)$ . To the best of our knowledge, this is the first application of the RIC framework to the construction of a Voronoi diagram with disconnected regions and disconnected bisectors.

The Hausdorff Voronoi diagram finds direct applications in Very Large Scale Integration (VLSI) circuit design, see [12] and references there in, and possibly to other networks embedded in the plane, such as transportation networks. It has been used by the semiconductor industry to efficiently estimate the *critical area* of a VLSI layout for various types of *open faults*, see e.g., [11,12,4], where critical area is a measure reflecting the sensitivity of the design to random de-

<sup>1</sup> We use term “range” instead of a more usual “region” to avoid confusion with the Voronoi regions.

fects during manufacturing. In a VLSI layout, clusters can indeed be crossing, however, their number of crossings is typically small. For more information on generalized Voronoi diagrams see, e.g., the book of Aurenhammer et al. [1].

## 2 Preliminaries

Let  $F$  be a family of  $k$  clusters of points in the plane, and let  $n$  be the total number of points in  $F$ . No two clusters have a common point. For simplicity of presentation, we follow a general position assumption that no four points lie on the same circle. We also assume that no cluster encloses another in its convex hull, as the outer hull would not be relevant to the Hausdorff diagram.

The *farthest Voronoi diagram* of a cluster  $C$ ,  $FVD(C)$ , is a partitioning of  $\mathbb{R}^2$  into regions where the *farthest Voronoi region* of a point  $c \in C$  is  $\text{freg}_C(c) = \{t \mid \forall c' \in C \setminus \{c\}: d(t, c) > d(t, c')\}$ . The graph structure of  $FVD(C)$  is a tree  $\mathcal{T}(C) = \mathbb{R}^2 \setminus \bigcup_{c \in C} \text{freg}_C(c)$ . We assume that  $\mathcal{T}(C)$  is rooted at a point at infinity along an unbounded Voronoi edge. If  $C = \{c\}$ , let  $\mathcal{T}(C) = c$ .

The *Hausdorff Voronoi diagram*, for brevity  $HVD(F)$ , is a partitioning of  $\mathbb{R}^2$  into regions, where the *Hausdorff Voronoi region* of a cluster  $C \in F$  is  $\text{hreg}_F(C) = \{p \mid \forall C' \in F \setminus \{C\}: d_f(p, C) < d_f(p, C')\}$ . The *Hausdorff Voronoi region* of a point  $c \in C$  is  $\text{hreg}_F(c) = \text{hreg}_F(C) \cap \text{freg}_C(c)$ .

Fig. 1 shows the Hausdorff Voronoi diagram of a family of four clusters. The convex hulls of the clusters are illustrated in grey lines. Solid lines indicate the borders of the Voronoi regions of individual clusters; the dashed lines indicate the finer subdivision of a Hausdorff Voronoi region  $\text{hreg}_F(C)$  into  $\text{hreg}_F(c)$ ,  $c \in C$ . Note that only points on the convex hull of a cluster may have non-empty regions, thus, we assume that each cluster equals its convex hull.

For two clusters  $C, P \in F$ , their Hausdorff bisector  $\text{b}_h(C, P) = \{y \mid d_f(y, C) = d_f(y, P)\}$  is a subgraph of  $\mathcal{T}(C \cup P)$ . It consists of one (if  $P, Q$  are non-crossing) or more (if  $P, Q$  are crossing) unbounded polygonal chains. Each vertex of  $\text{b}_h(C, P)$  is the center of a circle passing through two points of one cluster and one point of another that entirely encloses  $P$  and  $Q$ . These vertices are called *mixed*.

**Definition 2.** A vertex on the bisector  $\text{b}_h(C, P)$ , induced by two points  $c_i, c_j \in C$  and a point  $p_l \in P$  is called *crossing*, if there is a diagonal  $p_l p_r$  of  $P$  that crosses the diagonal  $c_i c_j$  of  $C$ , and all points  $c_i, c_j, p_l, p_r$  are on  $\text{conv } C \cup P$ . The total number of crossing vertices along the bisectors of all pairs of clusters is the number of crossings and is denoted by  $m$ .

The Hausdorff Voronoi diagram contains three types of vertices [13]: (1) *pure* Voronoi vertices, which are equidistant to three clusters; (2) *mixed* Voronoi vertices, which are equidistant to three points of two clusters; and (3) *farthest* Voronoi vertices, which are equidistant to three points of one cluster. The mixed vertices, which are induced by two points of cluster  $C$  (and one point of another cluster), are called *C-mixed* vertices. The Hausdorff Voronoi edges are polygonal lines (portions of Hausdorff bisectors) that connect pure Voronoi vertices and separate the Voronoi regions of different clusters (see e.g., the solid lines in

Fig. 1a). Mixed Voronoi vertices are the breakpoints of these polygonal lines. They are characterized as crossing or non-crossing according to Def. 2.

*Property 1 ([13]).* Each face of a (non-empty) region  $\text{hreg}_F(C)$  intersects  $\mathcal{T}(C)$  in one non-empty connected component. This component is delimited by  $C$ -mixed vertices.

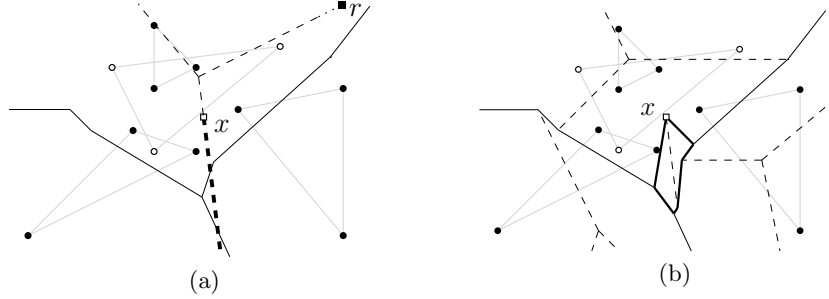
Unless stated otherwise, we use a refinement of the Hausdorff Voronoi diagram as derived by the *visibility-based* decomposition [14] of each region  $\text{hreg}_F(p)$  (see Fig. 1c): for each vertex  $v$  on  $\partial\text{hreg}_F(p)$  draw the line segment  $pv$ , as restricted within  $\text{hreg}_F(p)$ . In Fig. 1c, the edges of the visibility-based decomposition of  $\text{hreg}_F(p)$  are shown in bold. Let  $f$  be a face of  $\text{hreg}_F(p)$ , where  $\text{hreg}_F(p)$  is refined by the visibility-based decomposition. We often refer to point  $p$  as the *owner* of  $f$ .

**Observation 1** *A face  $f$  of  $\text{hreg}_F(p)$ ,  $p \in P$ , borders the regions of at most three other clusters in  $F \setminus \{P\}$ . These are exactly the clusters that, together with  $P$ , induce the Voronoi vertices delimiting the Hausdorff edge of  $f$ .*

Similarly to [5], we often answer *segment queries* on edges of the farthest Voronoi diagram of a cluster  $P$ . These queries can be answered in  $O(\log |P|)$  time using the *centroid decomposition* of  $\text{FVD}(P)$ , see [5]. The segment query is defined as follows:

*Segment Query.* Given  $\text{FVD}(P)$  and a segment  $uv \subset \mathcal{T}(C)$ , such that  $d_f(u, C) < d_f(u, P)$  and  $d_f(v, C) > d_f(v, P)$ ,  $P, C \in F$ , find the point  $x \in uv$  that is equidistant to  $C$  and  $P$ .

The centroid decomposition of  $\text{FVD}(P)$  is obtained by recursively breaking  $\mathcal{T}(P)$  into subtrees of its *centroid*.<sup>2</sup>



**Fig. 2.** Insertion of a cluster  $C$  (filled disks): (a)  $\mathcal{T}(C)$  (dashed) rooted at  $r$ , its active subtree (bold); (b) the HVD after the insertion:  $x$  is the root of the active subtree of  $\mathcal{T}(C)$ .

<sup>2</sup> Any tree with  $h$  vertices has a vertex called *centroid*, removal of which decomposes the tree into subtrees with at most  $h/2$  vertices each.

### 2.1 Overview of the RIC framework [6,7]

The RIC technique to compute a Voronoi diagram, inserts sites (or objects) one by one in random order, each time recomputing the target diagram. The diagram is viewed as a collection of *ranges*, *defined* and without *conflicts* with respect to the set of sites inserted so far. Each range must be defined by constant number of objects. To update the diagram efficiently, a *conflict* or *history* graph is maintained.

The conflict graph is a bipartite graph where one group of nodes correspond to the ranges of the diagram of sites inserted so far, and the other group of nodes correspond to sites that are not yet inserted. A range and a site are connected by an arc in the conflict graph if and only if they are in conflict. A RIC algorithm that uses a conflict graph is efficient if the following *update condition* is satisfied at each incremental step: (1) Updating the set of ranges defined and without conflict over the current subset requires time proportional to the number of ranges killed or created during this step; and (2) Updating the conflict graph requires time proportional to the number of arcs added or removed during this step.

The history graph is a directed acyclic graph, whose nodes are the ranges that have been created so far. The ranges of the current diagram are leaves, and parent(s) of a range are those ranges that are deleted at the step of its creation, and who intersect it. The *update condition* for an efficient RIC algorithm using a history graph is: (1) the existence of a conflict between a range and a site can be tested in  $O(1)$  time; (2) nodes of the history graph have bounded outdegree; (3) at each step updating the history graph can be done in time proportional to the number of ranges killed and created.

If the update conditions are satisfied then the expected time and space complexity of the RIC algorithm is as stated in [3, Theorem 5.2.3].

**Theorem 1 (Theorem 5.2.3 [3]).** *Let  $f_0(r)$  be the expected number of ranges in the target structure of the random sample of  $r$  objects. Then: (1) Expected number of ranges created during the whole algorithm is  $O\left(\sum_{r=1}^k (f_0(r)/r)\right)$ . (2) If the update conditions are satisfied, then the expected time and space complexity of the algorithm is  $O\left(k \sum_{r=1}^k (f_0(r)/r^2)\right)$ .*

## 3 Construction of HVD( $F$ ) for non-crossing clusters

Let the clusters in  $F$  be pairwise non-crossing. Then the Hausdorff Voronoi regions are connected and the combinatorial complexity of the diagram is  $O(n)$ . We first introduce some notation.

Let  $S \subset F$ , and  $C \in F \setminus S$ . Let  $\mathcal{T}_a(C, S)$  be the subtree of  $\mathcal{T}(C)$  that is relevant to the insertion of  $C$  into HVD( $S$ ), called the *active subtree* of  $\mathcal{T}(C)$ , defined as follows. Let  $x$  be the first point in  $\text{hreg}_{S \cup \{C\}}(C)$  that is encountered

as we traverse  $\mathcal{T}(C)$  starting from its root; point  $x$  must be a mixed vertex.  $\mathcal{T}_a(C, S)$ , is the subtree of  $\mathcal{T}(C)$  rooted at  $x$ .

Figs. 2a and 2b illustrate  $\text{HVD}(S)$  and  $\text{HVD}(S \cup \{C\})$  respectively. The points of  $C$  are shown as unfilled circles. Fig. 2a also illustrates  $\mathcal{T}(C)$  in dashed lines, superimposed on  $\text{HVD}(S)$ ;  $\mathcal{T}(C)$  is rooted at  $r$ . Point  $x$  is the root of  $\mathcal{T}_a(C, S)$ , which is shown in bold. Fig. 2b shows the boundary of  $\text{hreg}_{S \cup \{C\}}(C)$  in bold lines.

Property 1 together with the fact that Voronoi regions of non-crossing clusters are connected, imply the following property, which is the basis for our algorithm.

*Property 2.* For a family  $F$  of non-crossing clusters, a cluster  $C$  has a non-empty Voronoi region in  $\text{HVD}(S \cup \{C\})$  if and only if the active subtree  $\mathcal{T}_a(C, S)$  is non-empty. Moreover, the root of  $\mathcal{T}_a(C, S)$  is a  $C$ -mixed vertex of  $\text{HVD}(S \cup \{C\})$  (possibly a vertex at infinity).

This observation suggests that once the root of  $\mathcal{T}_a(C, S)$  is determined, cluster  $C$  can easily be inserted into  $\text{HVD}(S)$ . Thus, the crux of our problem becomes to efficiently maintain the root of  $\mathcal{T}_a(C, S)$  for every  $C$  in  $F \setminus S$ .

### 3.1 Objects, ranges and conflicts

We formulate the problem of computing  $\text{HVD}(F)$  in terms of *objects*, *ranges* and *conflicts*. Objects are the clusters in  $F$ . Ranges correspond to faces of  $\text{HVD}(S)$  as refined by the visibility-based decomposition. A range corresponding to face  $f$ ,  $f \subset \text{hreg}_S(P)$ , is said to be defined by  $P$  and the additional clusters in  $S$  whose Voronoi regions border  $f$ . These are at most four clusters (see Observation 1), which allows us to apply the RIC framework. We use the same notation to denote both a face of  $\text{HVD}(S)$  and its corresponding range.

**Definition 3 (conflict for non-crossing clusters).** A range  $f$  is in conflict with a cluster  $C \in F \setminus S$  if  $\mathcal{T}_a(C, S)$  is not empty and its root  $x$  lies in  $f$ . A conflict is a triple  $(f, x, C)$ . The list of conflicts of range  $f$  is denoted as  $\mathcal{L}(f)$ .

Each range  $f \subset \text{hreg}_S(p)$  stores a pointer to its owner  $p$  and a pointer to the list of its conflicts  $\mathcal{L}(f)$ . The following is a direct implication of Property 2.

**Corollary 1.** Each cluster  $C \in F \setminus S$  has at most one conflict. If a cluster  $C$  has no conflicts, then  $\text{hreg}_{S \cup \{C\}}(C) = \emptyset$ , and thus,  $\text{hreg}_F(C) = \emptyset$ .

Clusters with zero conflicts are ignored by our algorithm.

### 3.2 Insertion of a cluster, variant with the conflict graph

Suppose that  $\text{HVD}(S)$ ,  $S \subset F$  has been constructed together with its conflict graph. The algorithm to insert a cluster  $C \in F \setminus S$  in  $\text{HVD}(S)$  is given in Fig. 3 as pseudocode.

To update  $\text{HVD}(S)$  after the insertion of  $C$ , we trace the boundary of  $\text{hreg}_{S \cup \{C\}}(C)$  starting at the root of  $\mathcal{T}_a(C, S)$ , as described in [14]. To update

the conflict graph, for each deleted range  $f$ ,  $f \subset \text{hreg}_S(p)$ , we delete its conflicts, and create new conflicts in place of the deleted ones. Let  $(f, y, Q)$  be a conflict of  $f$ . If  $\mathcal{T}_a(Q, S \cup \{C\})$  is the same as  $\mathcal{T}_a(Q, S)$ , then we find a new range that contains the root  $y$  of  $\mathcal{T}_a(Q, S \cup \{C\})$  (Line 9 in Fig. 3), by performing a binary search on the edges of the visibility-based decomposition of  $\text{hreg}_{S \cup \{C\}}(p)$  ordered as they are seen from  $p$ . Otherwise, we first find the root  $z$  of  $\mathcal{T}_a(Q, S \cup \{C\})$ , and then we find the range of  $\text{HVD}(S)$  containing  $z$ . Using the (updated) root of the active subtree and the range containing it, we create the new conflict of  $Q$ . Line 15 expands as follows: We find the owner  $c \in C$  of the range  $h$ , such that  $z \in h$ , by performing a point location for  $z$  in  $\text{FVD}(C)$ . Then, the actual range  $h$  can be found by binary search as in Line 9.

**Algorithm** *Insert-NonCrossing*

(\* Inserts a cluster  $C$  in  $\text{HVD}(S)$ ; updates the conflict graph \*)

1. Let  $(g, x, C)$  be the conflict of  $C$ .
2. Trace the region  $\text{hreg}_{S \cup \{C\}}(C)$  starting from  $x$ .
3. Update  $\text{HVD}(S)$  to  $\text{HVD}(S \cup \{C\})$ .
4. **for**  $f \in \text{HVD}(S) \setminus \text{HVD}(S \cup \{C\})$  **do**
5.     Let  $p$  be the owner of  $f$ .
6.     **for** each conflict  $(f, y, Q) \in \mathcal{L}(f)$  **do**
7.         Discard the conflict  $(f, y, Q)$ .
8.         **if**  $d_f(y, p) < d_f(y, C)$  **then**
9.             Locate the range  $f' \subset \text{hreg}_{S \cup \{C\}}(p)$  that contains  $y$ .
10.            Create conflict  $(f', y, Q)$ .
11.         **else**
12.             Search for an edge  $uv$  in  $\mathcal{T}_a(Q, S)$  such that  $d_f(u, Q) > d_f(u, C)$  and  $d_f(v, Q) < d_f(v, C)$ .
13.             **if**  $uv$  is found **then**
14.                 Perform a segment query for  $uv$  in  $\text{FVD}(C)$  to find root  $z$  of  $\mathcal{T}_a(Q, S \cup \{C\})$ .
15.                 Locate the range  $h \subset \text{hreg}_{S \cup \{C\}}(C)$  that contains  $z$ .
16.                 Create conflict  $(h, z, Q)$

**Fig. 3.** Algorithm to insert cluster  $C$ ; case of non-crossing clusters.

**Lemma 1.** *The algorithm Insert-NonCrossing (Fig. 3) is correct.*

*Proof.* In Line 8, it is enough to compare distance from  $y$  to only  $p$  and  $C$  because no other cluster may become the closest to  $y$  as a result of inserting  $C$ . The correctness of Lines 9–10, i.e. the case when  $d_f(y, p) < d_f(y, C)$ , is easy to see. Suppose that  $d_f(y, C) < d_f(y, Q) = d_f(y, p)$ , and that  $\mathcal{T}_a(Q, S \cup \{C\}) \neq \emptyset$ . In this case  $y$  is no longer a part of the (updated) diagram  $\text{HVD}(S \cup \{C\})$ . Clearly,  $\mathcal{T}_a(Q, S \cup \{C\})$  is a subtree of  $\mathcal{T}_a(Q, S)$ , thus, there is exactly one edge  $uv$  of  $\mathcal{T}_a(Q, S)$  such that  $u \notin \mathcal{T}_a(Q, S \cup \{C\})$  and  $v \in \mathcal{T}_a(Q, S \cup \{C\})$ . Edge  $uv$  satisfies the condition of Line 12 by definition of an active subtree. The root of an active subtree cannot coincide with a vertex of  $\mathcal{T}(C)$  due to the general position assumption.



### 3.3 Complexity analysis for the conflict graph

We now analyze the time and space complexity of our randomized incremental algorithm. Consider a random permutation  $\{C_1, \dots, C_k\}$  of the input family  $F$ , and the sequence  $\{F_0, \dots, F_k\}$ , where  $F_0 = \emptyset$ ,  $F_i = F_{i-1} \cup \{C_i\}$  and  $F_k = F$ . At step  $i$  we insert cluster  $C_i$ , following the procedure described in Section 3.2. Corollary 1 directly implies the following.

**Lemma 2.** *The number of arcs in the conflict graph at any step is  $O(k)$ .*

**Lemma 3.** *Updating the conflict graph at step  $i$  of the algorithm requires  $O(\log n(N_i + R_i))$  time, where  $N_i$  is the total number of edges dropped out of the active subtrees of clusters in  $F \setminus F_i$  at this step;  $R_i$  is the total number of conflicts deleted at this step.*

*Proof.* Updating the conflict graph corresponds to two nested for-loops in Lines 4–16 of the algorithm in Fig. 3. Clearly the inner loop (Lines 6–15) is performed  $O(R_i)$  times. Inside this loop, the breadth-first search is performed that spends  $O(\log n)$  time per visited edge. By Corollary 1, one active subtree is considered at most once. All the visited edges, except the last one, are dropped out of the respective active subtree. It remains to show, that, except for the breadth-first search, the rest of the work in any execution of the inner loop requires  $O(\log n)$  time. Indeed, it is a point location in Line 8, a segment query in  $FVD(C_i)$  in Line 13, and a binary search in Line 9 or Line 15.

**Lemma 4.** *The expected total number of conflicts deleted at step  $i$  of the randomized incremental algorithm is  $O(k/i) + D_i$ , where  $D_i$  is the number of clusters in  $F \setminus F_i$  that used to have a conflict until step  $i$ , and do not have it any more.*

*Proof.* Each conflict deleted at step  $i$  either induces a conflict with a (new) range in  $HVD(F_i) \setminus HVD(F_{i-1})$ , or the corresponding cluster is counted by  $D_i$ . Each cluster is counted at most once by  $D_i$  due to Corollary 1. Thus, the total number of conflicts deleted at step  $i$  is  $D_i$  plus the total number of conflicts of ranges inserted at step  $i$ . Applying backwards analysis, it is easy to see that the expectation of the latter number is  $O(k/i)$ .

**Theorem 2.** *The randomized incremental construction algorithm to compute the Hausdorff Voronoi diagram of  $k$  non-crossing clusters of total complexity  $n$ , requires  $O(n)$  space, and expected  $O(n \log n + k \log n \log k)$  time.*

*Proof.* The expected total number of ranges created and deleted during the algorithm is  $O(n)$ , see [5, Theorem 2]. Updating the Hausdorff Voronoi diagram can be done in time proportional to this number using the tracing described in [14]. Now we will analyze only the work to update the conflict graph. By Lemma 3, the total time required for this task is  $\sum_{i=1}^k O(\log n(N_i + R_i))$ . An edge of  $\mathcal{T}(C)$  of any cluster  $C \in F$  is dropped out from the active subtree at most once. The total number of edges in the farthest Voronoi diagrams of all clusters

in  $F$  is  $O(n)$ . Thus the above sum is proportional to  $O(\log n)(n + \sum_{i=1}^k R_i)$ . The expectation of this number is, by Lemma 4,  $O(n + \sum_{i=1}^k (k/i + D_i \log n))$ . Note that  $\sum_{i=1}^k D_i \leq k$ , since the active subtree of a cluster can become empty at most once. The claimed time complexity follows.

The space requirement at any step is proportional to the combinatorial complexity of the Hausdorff Voronoi diagram, which is  $O(n)$ , plus the total number of arcs of the conflict graph at this step, which is at most  $k$  by Lemma 2. Hence the claimed  $O(n)$  bound holds.

### 3.4 Adapting the algorithm to using a history graph

Let  $\mathcal{H}(F_i)$  denote the history graph that has been computed at step  $i$  of the incremental algorithm.  $\mathcal{H}(F_0)$  is a single node that corresponds to the whole  $\mathbb{R}^2$ . For  $i \in \{1, \dots, k\}$ ,  $\mathcal{H}(F_i)$  consists of all nodes and arcs of  $\mathcal{H}(F_{i-1})$  and in addition it contains the following: (i) A node for each new range in  $\text{HVD}(F_i) \setminus \text{HVD}(F_{i-1})$ . These nodes are called the *nodes of level  $i$* . (ii) An arc connecting a deleted range  $f \in \text{HVD}(F_{i-1}) \setminus \text{HVD}(F_i)$  to every new range  $f' \in \text{HVD}(F_i) \setminus \text{HVD}(F_{i-1})$  such that  $f'$  intersects  $f$ .

Suppose that  $\text{HVD}(F_{i-1})$  and  $\mathcal{H}(F_{i-1})$  are already computed. To insert the next cluster  $C_i$ , we traverse  $\mathcal{H}(F_{i-1})$  from root to a leaf. Simultaneously, we move in  $\mathcal{T}(C_i)$ , keeping track of the root  $x$  of the active subtree  $\mathcal{T}_a(C_i, F_j)$  at the current level  $j$  of  $\mathcal{H}(F_{i-1})$ . When we reach a leaf of  $\mathcal{H}(F_{i-1})$ , we trace the boundary of the Voronoi region  $\text{hreg}_{F_i}(C_i)$ , starting at the root  $x$  of  $\mathcal{T}_a(C_i, F_i)$ , and update  $\mathcal{H}(F_{i-1})$  to become  $\mathcal{H}(F_i)$ .

In more detail, the procedure for level  $j$  is as follows: Let  $f$  be the face in  $\text{HVD}(F_j)$  that contains  $x$ . Suppose that  $f$  is deleted at step  $\ell$ . If  $d_f(x, C_i) < d_f(x, C_\ell)$ , we search for the child  $f'$  of  $f$ , that has the same owner as  $f$ , and contains  $x$ ; we move to the level  $\ell$ , keeping  $x$  intact, and updating its face to be  $f'$ . Else we search for the root  $z$  of the (new) active subtree  $\mathcal{T}_a(C_i, F_\ell \cup \{C_i\})$  (the procedure to do this is the same as for the conflict graph, see Sec. 3.2). If  $z$  is found, we move to level  $\ell$ , replace  $x$  by  $z$  and the face  $f$  by  $f' \subset \text{hreg}_{F_\ell}(C_\ell)$  that contains  $z$ . If  $z$  is not found, the active subtree  $\text{hreg}_{F_i}(C_i) = \emptyset$ .

Updating the history graph during step  $i$  takes time  $O(\log n(N_i + K_i))$ , where  $N_i$  is the number of edges of  $\mathcal{T}(C_i)$  that do not belong to the active subtree  $\mathcal{T}_a(C_i, F_i)$ , and thus, they are eliminated by the breath-first search.  $K_i$  is the number of clusters in the sequence  $\{C_1, \dots, C_{i-1}\}$  that change the root of the active subtree as we move in the history graph level by level. The expectation of  $K_i$  is  $O(\log i)$ . Summing over all  $k$  steps gives us  $O(k \log k)$ . The total expected running time of the algorithm using the history graph is thus,  $O(n \log n + k \log n \log k)$ .

## 4 Computing $\text{HVD}(F)$ for arbitrary clusters of points

In this section we drop the assumption that clusters in the input family  $F$  are pairwise non-crossing. This raises a major difficulty that Voronoi regions may be disconnected in multiple faces. Using the simple definition of conflict from Sec. 3.1 in this situation does not guarantee that the diagram will be computed correctly. Thus we give a new definition for a conflict. We first recall the general setting of the algorithm, that is the same as in Sec. 3.1, then we give a new definition of the conflict and a procedure to update the conflict graph, and analyze the complexity of the resulting algorithm.

Let  $C_1, \dots, C_k$  be a random permutation of clusters in  $F$ . We incrementally compute  $\text{HVD}(F_i)$ ,  $i = 1, \dots, k$ , where  $F_i = \{C_1, C_2, \dots, C_i\}$ . At each step  $i$ , cluster  $C_i$  is inserted in  $\text{HVD}(F_{i-1})$ . To perform the insertion efficiently, we maintain the conflict graph. Recall from Section 2.1, that the conflict graph at the moment right before the insertion of the cluster  $C_i$  is a bipartite graph whose one group of nodes correspond to all the ranges of  $\text{HVD}(F_{i-1})$ , and the other group of nodes correspond to clusters in  $F \setminus F_{i-1}$ . A range and a cluster are connected by an arc in the conflict graph if and only if they are in conflict. Similarly to Sec. 3.1, ranges are the faces (of the visibility-based decomposition) of  $\text{HVD}(F_{i-1})$ . Conflicts are defined as follows (differently from Sec. 3.1):

**Definition 4 (Conflict for arbitrary clusters).** *Let  $f$  be a range of  $\text{HVD}(F_{i-1})$ ,  $f \subset \text{hreg}_{F_{i-1}}(p)$  for a point  $p, p \in P$ . Range  $f$  is in conflict with a cluster  $Q \in F \setminus F_{i-1}$ , if  $f$  intersects the boundary of  $\text{hreg}_{F_{i-1} \cup \{Q\}}(Q)$ . A conflict is a 4-tuple  $(f, p, Q, v)$ , where  $v$  is the vertex list of the conflict, defined as the list of the vertices and endpoints of  $\text{b}_h(P, Q) \cap \bar{f}$  in the order in which they appear on  $\text{b}_h(P, Q)$ .*

Inserting of cluster  $C_i$  consists of (1) computing  $\text{HVD}(F_i)$  from  $\text{HVD}(F_{i-1})$  using the conflict graph, and (2) updating the conflict graph.

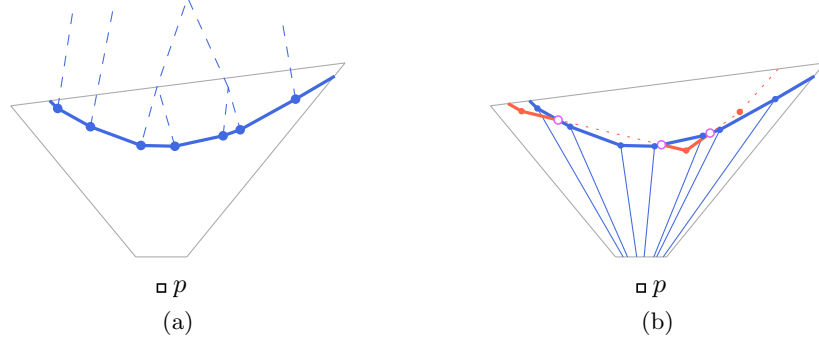
Step (1) is performed in the same way as in the case of non-crossing clusters: the boundary of the new face is traced starting at the  $C_i$ -mixed vertices on this boundary. All the necessary  $C_i$ -mixed vertices are elements of the vertex lists of the conflicts of  $C_i$ .

During step (2), the conflict graph changes as follows: For each deleted range  $f$ , all the conflicts of  $f$  are deleted, and new conflicts are created between the new ranges that intersect  $f$  and clusters that were in conflict with  $f$ .

Below we focus on the efficient procedure to update the conflict graph after the insertion of cluster  $C_i$  (step (2)). The technical challenge of this task is caused by the non-constant complexity of the Hausdorff bisector between two clusters, and subsequently the non-constant complexity of the vertex lists of the conflicts deleted and created. In addition, the number of new ranges that intersect the deleted  $f$  is non-constant.

We are updating the conflict graph in such way that all the work done can be charged to conflicts that are created or deleted, to the vertices that are deleted from the vertex lists of conflicts and do not appear in the vertex lists of any new

conflicts, and to the total size for the vertex lists of the conflicts of the ranges of  $\text{hreg}_{F_i}(C_i)$ .



**Fig. 4.** A (deleted) range  $f$  of  $\text{hreg}_{F_{i-1}}(p)$  (grey); (a) Chain  $\mathbf{b}_h(p, C_i) \cap \bar{f}$  and the adjacent portion of  $\mathcal{T}(C_i)$ ; (b) (New) ranges of  $\text{hreg}_{F_i}(p)$  that intersect  $f$  (bounded by blue solid lines); The chain  $\mathbf{b}_h(p, Q) \cap \bar{f}$  (red), where  $Q$  is a cluster in conflict with  $f$ . The portions of this chain that are closer to  $p$  than to  $C_i$  (red, bold), and the ones closer to  $C_i$  than to  $p$  (red, dotted).

For each cluster  $Q$  in conflict with any of the ranges of  $\text{HVD}(F_{i-1})$  deleted while  $C_i$  is inserted, we need to create all the conflicts:

- (1) between  $Q$  and the new ranges of  $\text{hreg}_{F_i}(C_i)$ , and
- (2) between  $Q$  and the new ranges of the clusters in  $F_{i-1}$ .

The ranges from item (2) are portions of the ones of  $\text{HVD}(F_{i-1})$ , which were clipped and subdivided by insertion of  $\text{hreg}_{F_i}(C_i)$  (see the ranges bounded by blue lines in Fig. 4b).

To create the conflicts of type (1), we trace  $\mathbf{b}_h(C_i, Q)$  in  $\text{hreg}_{F_i}(C_i)$ .

Conflicts of type (2) are created as follows. For a deleted range  $f$  of  $\text{hreg}_{F_{i-1}}(p)$ , we need to create all the conflicts between  $Q$  and the new ranges of  $\text{hreg}_{F_i}(p)$  that intersect  $f$  (see solid red lines in Fig. 4b). We could trace  $\mathbf{b}_h(p, Q) \cap f$  in  $\text{hreg}_{F_i}(p)$ . However, if we would do this by visiting  $\mathbf{b}_h(p, Q)$ , each such vertex might be visited many times during the course of the algorithm. Instead of visiting these vertices, we consider rays from  $p$  to the vertices of  $\mathbf{b}_h(p, C_i) \cap f$ . These rays subdivide the vertices of  $\mathbf{b}_h(p, Q)$  among the vertex lists of the new ranges of type (2). To get from one component of  $\mathbf{b}_h(p, Q) \cap \text{hreg}_{F_i}(p) \cap f$  to another one, we trace the deleted part of  $\mathbf{b}_h(p, Q) \cap f$  in  $\text{hreg}_{F_i}(C_i)$ .

We give the details in the following.

**Lemma 5.** *Updating the conflict graph after the insertion of cluster  $C_i$  can be done in time  $O((A(C_i) + L(C_i) + V(C_i)) \log n)$ , where  $A(C_i)$  is the number of*

conflicts created and deleted,  $L(C_i)$  is the total number of mixed vertices discarded from the vertex lists of conflicts that do not appear in the vertex lists of new conflicts, and  $V(C_i)$  is the total number of vertices in the vertex lists of all conflicts of all ranges of  $\text{hreg}_{F_i}(C_i)$ .

*Proof.* To create the conflicts of type (1), we only need one endpoint of every edge of  $\text{HVD}(F_i \cup \{Q\})$  induced by  $C_i$  and  $Q$ . Starting from each such endpoint, we trace  $\mathbf{b}_h(Q, C_i)$  inside  $\text{hreg}_{F_i}(C_i)$  until it reaches the boundary of  $\text{hreg}_{F_i}(C_i)$ . While tracing, we create conflicts between  $Q$  and the ranges of  $\text{hreg}_{F_i}(C_i)$  intersected by the traced portion of  $\mathbf{b}_h(Q, C_i)$ . The vertex lists of these conflicts contain exactly the vertices of traced portion of  $\mathbf{b}_h(Q, C_i)$  inside the corresponding ranges, and intersection points of  $\mathbf{b}_h(Q, C_i)$  with the boundary of these ranges. The number of vertices of  $\mathbf{b}_h(Q, C_i)$  visited during such tracing is counted by  $V(C_i)$ . for each  $Q \in F \setminus F_i$ , the above procedure discovers all the edges of  $\text{HVD}(F_i \cup Q)$  induced by  $C_i$  and  $Q$ : If it would be not the case, then  $Q$  would have a face in  $\text{HVD}(F_i \cup Q)$  bounded solely by the edges induced by  $C_i$  and  $Q$ , i.e., a face that lies inside the region of another cluster, which is not possible.

Now we create the conflicts of type (2). As a byproduct, we also find all the edge endpoints that are required by the above procedure of creating the conflicts of type (1). We do this by iterating over all the conflicts of the deleted ranges. Let  $(f, p, Q, v)$  be such conflict, where  $f$  is a deleted range,  $f \subset \text{hreg}_{F_{i-1}}(p)$ ,  $p \in P$ ,  $P \in F_{i-1}$ , see Fig. 4. Consider  $\mathbf{b}_h(P, Q) \in \bar{f}$ . It is a sequence of polygonal curves, which can be retrieved from the vertex list  $v$  (in Fig. 4b, it is one red polygonal curve). Our task is to detect which of the new ranges owned by  $p$  and intersected by  $f$  are in conflict with  $Q$ , and create the corresponding conflicts. This is equivalent to determining the portions of  $\mathbf{b}_h(P, Q) \in \bar{f}$  that are closer to  $p$  than to  $C_i$  (respectively, the solid and the dotted portions of the red chain in Fig. 4). The endpoints of these portions, that are not endpoints of  $\mathbf{b}_h(P, Q) \in \bar{f}$ , are intersection between  $\mathbf{b}_h(P, Q)$  and  $\mathbf{b}_h(P, C_i)$  (marked by empty violet circles in Fig. 4). These intersections are exactly the endpoints of edges, required for detecting the conflicts of type (1) (see the previous paragraph).

Unlike the case of conflicts of type (1), we cannot afford tracing  $\mathbf{b}_h(P, Q) \in \bar{f}$  vertex by vertex, since its vertices that are closer to  $p$  than to  $C_i$  are counted neither by  $L(C_i)$  nor by  $V(C_i)$ . However we can distribute the list of these vertices among the new ranges owned by  $p$  and intersected by  $f$ , by locating the rays connecting  $p$  with the vertices of  $\mathbf{b}_h(p, C_i) \cap \bar{f}$ , since these rays are exactly the ones bounding the ranges under consideration (see thin solid blue lines in Fig. 4b). To this aim, instead of storing vertex lists with conflicts, we store all of them (gathered) with the clusters. That is, for each cluster  $Q \in F \setminus F_i$ , we store with  $Q$  the vertices of each edge  $e$  that bounds a face of  $\text{hreg}_{F_i \cup \{Q\}}(Q)$ , in the order in which they appear on that boundary. Each conflict of  $Q$  stores a pair of pointers delimiting the portion of  $\mathbf{b}_h(P, Q)$  that corresponds to its vertex list. Thus, in order to create a conflict between  $Q$  and a new range owned by  $p$  (a conflicts of type (2)), we perform two binary searches in such list stored with  $Q$ .

Recall that there can be several connected components of  $\mathbf{b}_h(P, Q)$  closer to  $p$  than to  $C_i$  (see the solid red lines in Fig. 4b). Each such component is processed

as described above. After one component is processed, the beginning of the next one must be found. We do this by tracing the portion of  $b_h(p, Q)$  that is closer to  $C_i$  than to  $p$  (the dotted red lines in Fig. 4b), until we again encounter another portion that is closer to  $p$  than to  $C_i$ . Since both  $b_h(p, Q)$  and  $b_h(p, C_i)$  are concave polygonal chains (with respect to  $p$ ), each such portion contains at least one vertex of  $b_h(p, Q)$ , thus all the portions are detected. The vertices visited during this tracing are the ones that get deleted from the vertex list of  $f$  and do not appear in new conflicts, and thus these vertices are counted in  $L(C_i)$ .

To visit one vertex  $v$  we spend  $O(\log n)$  time, since we need to compare distances from  $v$  to  $P$  and to  $C_i$ , which requires a point location in  $FVD(P)$  and in  $FVD(C_i)$ . The time required to create the conflicts of type (2) and to find the starting points for conflicts of type (1) is therefore  $O((A(C_i) + L(C_i)) \log n)$ . The time required to create the conflicts of type (1) is  $O((A(C_i) + V(C_i)) \log n)$ .

After all the new conflicts are created, for each deleted range  $f$ , we delete all the conflicts of  $f$ , which requires  $O(A(C_i))$  time.

**Lemma 6.** *At any step  $i$ , the total number of vertices in the vertex lists of all the conflicts is  $O(n + m + N_i)$ , where  $N_i$  is the number of conflicts in the current conflict graph.*

*Proof.* Consider a cluster  $Q \in F \setminus F_i$ . Members of the vertex lists of the conflicts of  $Q$  are: (1) the mixed vertices of  $HVD(F_i \cup \{Q\})$  that bound the region of  $Q$  in that diagram,  $\text{hreg}_{F_i \cup Q}(Q)$ , and (2) points that are not mixed vertices of  $HVD(F_i \cup \{Q\})$ . There are at most two points of type (2) per one conflict of  $Q$ . Indeed, consider a conflict  $(f, p, Q, v)$ . Points of type (2) related to this conflict are exactly the intersections between two polygonal chains,  $b_h(p, Q)$  and  $\partial f \setminus \mathcal{T}(P)$ , where  $p \in P$ . The first chain is concave (as seen from  $p$ ), and the second one is convex. Therefore the total number of points of type (2) is proportional to the number of conflicts of  $Q$ .

Now we bound the number of mixed vertices on the boundary of  $\text{hreg}_{F_i \cup \{Q\}}(Q)$  that are in the vertex list of the conflict  $(f, p, Q, V)$ , where  $p \in P$ . They are  $P$ -mixed and  $Q$ -mixed vertices. Since the boundary of  $f$  that is a portion of  $\mathcal{T}(P)$  is connected, there can be at most two non-crossing  $P$ -mixed vertices in  $V(f, Q)$ , all other vertices are crossing mixed vertices induced by  $P$  and  $Q$ . Thus the total number of mixed vertices bounding  $\text{hreg}_{F_i \cup \{Q\}}(Q)$  is  $O(|Q| + N_i + Cr(Q))$ , where  $Cr(Q)$  is the number of crossings between  $Q$  and all the clusters in  $F_i$ . The claim follows by summing up these numbers over all clusters  $Q$  in  $F \setminus F_i$ .

**Theorem 3.** *The Hausdorff Voronoi diagram of a family  $F$  of  $k$  clusters of total complexity  $n$  can be computed in  $O((m + n \log k) \log n)$  expected time and  $O(m + n \log k)$  expected space.*

*Proof.* We aim to apply Lemma 5, and in order to do that, we need to estimate the expectation of the total number of conflicts created during the course of the algorithm, and the expectation of the sum of  $L(C_i)$  and of  $V(C_i)$ , for  $i = 1, \dots, k$ . Additionally, we analyze the space required for the algorithm.

To analyze the expected total number of conflicts created during the course of the algorithm, we need to estimate the expected number of ranges, i.e., faces, in  $\text{HVD}(R)$ , where  $R$  is a random  $r$ -sample of  $F$ . The number of faces in a Hausdorff Voronoi diagram is proportional to the number of its mixed Voronoi vertices [13]. The number of non-crossing mixed vertices in  $\text{HVD}(R)$  is proportional to the total number of points in all clusters in  $R$  [13]. The expectation of the latter number is  $O(nr/k)$ : for each of  $n$  points in  $F$ , the probability that this point appears in  $R$  is the probability that its cluster appears in  $R$ , which is  $r/k$ .

For a crossing mixed vertex  $v$  induced by clusters  $P, Q \in F$ , the probability that  $v$  appears in  $\text{HVD}(R)$  is at most the probability that both  $P$  and  $Q$  appear in  $R$ , which is  $O(r^2/k^2)$ . Summing over all crossings of clusters in  $S$ , we have that the expected number of crossing mixed vertices in  $\text{HVD}(R)$  is  $O(mr^2/k^2)$ . Therefore, the expected number of ranges in  $\text{HVD}(R)$  is  $O(nr/k + mr^2/k^2)$ . Theorem 1(1) implies that the expected total number of conflicts created during the course of the algorithm is  $O(n \log k + m)$ .

The expected space complexity is  $O(n \log k + m)$ , implied by Theorem 1 and by Lemma 6, stating that the additional space required to store vertex lists of the conflicts at each step is  $O(n + m)$ .

It remains to bound the expectation of the sum of  $L(C_i)$  and of the sum of  $V(C_i)$ , for  $i = 1, \dots, k$ . We first note that since each deleted vertex was created at some point, the first sum is bounded by the second one. Thus we should bound the total number mixed vertices that appear in vertex lists of the conflicts during the course of the algorithm. First note, that we need to bound only the number of mixed vertices that appear in the vertex lists of the conflicts, since the total number of the ones that are not mixed vertices is proportional to the total number of conflicts created during the course of the algorithm, and thus  $O(n \log k + m)$ . The total number of all possible crossing mixed vertices is  $O(m)$ , and therefore we only need to bound the non-crossing mixed vertices.

For a cluster  $Q \in F$ , there are  $O(|Q|)$  non-crossing  $Q$ -mixed vertices at any step, at most one vertex per one edge of  $\mathcal{T}(Q)$ . For an edge  $e$  of  $\mathcal{T}(Q)$ , there can be up to  $k - 1$  possible  $Q$ -mixed vertices on  $e$ ; each such vertex  $v$  can be assigned weight equal to  $d_f(v, Q)$ . If a vertex  $v$  appears in the diagram at some step, any vertex  $v'$  with greater weight is guaranteed not to appear. The sequence of insertions of the clusters is a random permutation of  $F$ , and it corresponds to a random permutation of the sequence of the weights of mixed vertices along  $e$ . The ones that appear on the diagram are exactly the ones that change minimum in the partial sequence so far, and the expected number of such minimum changes is known to be  $O(\log k)$  [8]. The overall number of edges in  $\mathcal{T}(Q)$  for all  $Q \in F$  is  $O(n)$ , thus the total number of non-crossing vertices that appear during the course of the algorithm is  $O(n \log k)$ . Therefore the sum of  $V(C_i)$ , for  $i = 1, \dots, k$ , is  $O(n \log k + m)$ .

The claim now follows from Lemma 5.

## 5 A crossing-oblivious algorithm

We finally discuss how to proceed if it is not known whether the input family of clusters is crossing or non-crossing. Note that given a family of clusters, deciding whether it is crossing is difficult, since the clusters in the input set may be non-crossing, however their convex hulls may have a quadratic total number of intersections.

Our crossing-oblivious algorithm is a combination of the light algorithm for non-crossing clusters (Sec. 3) and the heavy algorithm for arbitrary clusters (Sec. 4). We start with the former algorithm and run it until we realize that the situation is too complex to be handled correctly by this algorithm. If this happens, we terminate the algorithm of Sec. 3, and run the one of Sec. 4. To detect the need to switch to the heavy algorithm, after each insertion of a cluster  $C_i$  we perform a check. The positive result of this check guarantees that the region of  $C_i$  in  $\text{HVD}(F_i)$  is connected, and thus it is computed correctly by the light algorithm. The negative result indicates that  $C_i$  has a crossing with some cluster which has been already inserted in the diagram. This means that the region of  $C_i$  in the Hausdorff Voronoi diagram of  $F_i$  may have more than one connected component. After the first negative check, we restart the computation of the diagram from scratch using the heavy algorithm of Sec 4. We can afford running the latter algorithm, since it is certain that the input set of clusters has crossings.

We proceed with the description and justification of the check. Its correctness is based on the following, which is a direct implication of the properties established in [13]:

*Property 3.* If  $\text{hreg}_{F_i}(C_i)$  has more than one connected components, then each of its connected components is incident to a crossing  $C_i$ -mixed vertex

The following lemma gives an efficient procedure for the check.

**Lemma 7.** *For a connected component  $f$  of  $\text{hreg}_{F_i}(C_i)$ , detecting whether there is a crossing  $C_i$ -mixed vertex on the boundary of  $f$  can be done in time  $O(|C| \log n)$ .*

*Proof.* Let  $v$  be a  $C_i$ -mixed vertex on the boundary of  $f$ , and let  $Q$  be the cluster, that together with  $C_i$  induces the vertex  $v$ . Vertex  $v$  breaks  $\mathcal{T}(C_i)$  in two (open) connected portions: the one that intersects  $f$ , and the one that does not intersect it. The former portion clearly contains points that are closer to  $C$  than to  $Q$ . Our task is to check whether the other portion contains such points as well. If this is the case, then  $C_i$  and  $Q$  are crossing, and  $v$  is a crossing mixed vertex. Otherwise,  $v$  is a non-crossing  $C_i$ -mixed vertex. The portions of  $\mathcal{T}(C_i)$ , that are to be checked because of different  $C_i$ -mixed vertices on the boundary of  $f$ , are disjoint. Thus it remains to show how to perform such check for each vertex in time proportional to the number of edges in the related portion of  $\mathcal{T}(C_i)$ , times  $O(\log n)$ .

Observe, that given two points  $u, v$  on an edge  $e$  of  $\mathcal{T}(C_i)$ , if both  $u$  and  $v$  are closer to  $Q$  than to  $C_i$ , then all the points on  $e$  between  $u$  and  $v$  are as well



closer to  $Q$  than to  $C$ . Indeed, consider the two closed disks  $D_u, D_v$  centered respectively at  $u$  and at  $v$  whose radii equal  $d_f(u, C)$  and  $d_f(v, C)$ . Since both  $u, v$  are closer to  $Q$  than to  $C$ , cluster  $Q \subset D_u \cap D_v$ . Further, the disk  $D_w$  centered at any point  $w \in e$  with radius  $d_f(w, C)$  contains  $D_u \cap D_v$ , and thus it contains  $Q$ , which means that  $w$  is closer to  $Q$  than to  $C$ .

By the above observation, it is enough to check all the vertices of the portion of  $\mathcal{T}(C)$  related to vertex  $v$ . If all of them are closer to  $Q$  than to  $C_i$ , then  $v$  is a non-crossing mixed vertex. Otherwise,  $v$  is crossing. A check for one vertex is a point location in FVD( $Q$ ), and thus it requires  $O(\log |Q|) = O(\log n)$  time.

We conclude.

**Theorem 4.** *Let  $F$  be a family of  $k$  clusters of total complexity  $n$ . There is an algorithm that computes  $\text{HVD}(F)$ . If the clusters in  $F$  are non-crossing, the algorithm requires  $O(n)$  space, and expected  $O(n \log n + k \log n \log k)$  time. If the clusters in  $F$  are crossing, the algorithm requires  $O((m + n \log k) \log n)$  expected time and  $O(m + n \log k)$  expected space, where  $m$  is the total number of crossings between pairs of clusters in  $F$ .*

## References

1. Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.
2. Cecilia Bohler and Rolf Klein. Abstract Voronoi diagrams with disconnected regions. *Int. J. Comput. Geometry Appl.*, 24(4):347–372, 2014.
3. Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge University Press, New York, NY, USA, 1998.
4. Voronoi CAA: Voronoi Critical Area Analysis. IBM VLSI CAD Tool, IBM Microelectronics Division, Burlington, VT. Distributed by Cadence. Patents: US6178539, US6317859, US7240306, US7752589, US7752580, US7143371, US20090125852.
5. Panagiotis Cheillaris, Elena Khramtcova, Stefan Langerman, and Evanthia Papadopoulou. A randomized incremental algorithm for the Hausdorff Voronoi diagram of non-crossing clusters. *Algorithmica*, 2016. DOI 10.1007/s00453-016-0118-y.
6. Kenneth Clarkson and Peter Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
7. Kenneth L. Clarkson, Kurt Mehlhorn, and Raimund Seidel. Four results on randomized incremental constructions. *Comput. Geom. Theory Appl.*, 3(4):185–212, 1993.
8. Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry – Algorithms and Applications*, 3rd ed. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
9. Frank Dehne, Anil Maheshwari, and Ryan Taylor. A coarse grained parallel algorithm for Hausdorff Voronoi diagrams. In *35th ICPP*, pages 497–504, 2006.
10. Herbert Edelsbrunner, Leonidas Guibas, and Micha Sharir. The upper envelope of piecewise linear functions: algorithms and applications. *Discrete Comput. Geom.*, 4:311–336, 1989.
11. E. Papadopoulou. Critical area computation for missing material defects in VLSI circuits. *IEEE Transactions on Computer-Aided Design*, 20(5):583–597, May 2001.

12. E. Papadopoulou. Net-aware critical area extraction for opens in VLSI circuits via higher-order Voronoi diagrams. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 30(5):704–717, 2011.
13. Evanthia Papadopoulou. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica*, 40(2):63–82, 2004.
14. Evanthia Papadopoulou and D. T. Lee. The Hausdorff Voronoi diagram of polygonal objects: a divide and conquer approach. *Int. J. Comput. Geom. Ap.*, 14(6):421–452, 2004.